Московский авиационный институт
(технический исследовательский университет)

Кафедра 403
“Электронно-вычислительные средства и информатика”

Шаблоны функций и классов.

Методическое пособие
и практические задания

Выполнила студентка группы № 40-102С
Артемьева А. А.

Руководитель ст. преподаватель каф. 403
Мальшаков Г. В.

г.Москва
2013 г.

Шаблоны (англ. *template*) — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).

Клише для печати денежных средств, тоже является шаблоном:

При программировании, с помощью магического заклинания **tamplate** мы можем превратить функцию либо класс в шаблон. Параметры шаблона – это типы данных обрабатываемые функцией. Параметры записываются в < > после ключевого слова **class** или **typename**.

У шаблона класса может быть несколько параметров-типов:

```
template <class T1, class T2, class T3>
class Container{//тело}; /* В шаблоне классов Container три абстриктных типа.
T1, T2, T3 – параметры шаблона. Они используются внутри класса в качестве
типов.*/
```

Ключевое слово **class** или **typename** должно предшествовать каждому параметру. Следующее объявление ошибочно:

```
template <typename T, U>    /* ошибка: перед Uотсутствует ключевое слово,
должно быть <typename T, class U> или <typename T, typename U>*/
class collection {//тело};
```

Имя параметра шаблона может встречаться в списке только один раз. Поэтому следующее объявление компилятор помечает как ошибку:

```
template <class Type, class Type> /* ошибка: в заголовке имя параметра шаблона
встречается несколько раз*/
class container{//тело};
```

Такое имя разрешается повторно использовать в объявлениях или определениях других шаблонов.

Параметр-константа шаблона представляет собой обычное объявление параметра. Шаблон класса **Buffer** имеет параметр-тип (Type), представляющий типы элементов, хранящихся в буфере, и параметр-константу (size), содержащий его размер:

```
template <class Type, int size>
class Buffer{//тело};
```

Шаблоны функций – это предписание, по которому компилятор создает функцию для заданных типов данных.

```
template < параметры_шаблона >
тип имя_функции ( параметры ) {//тело}
```

Пример:

```
template <class T>
void abs (T x) {return x<0? –x : x;}
```

Ограничения на параметры шаблона функции:

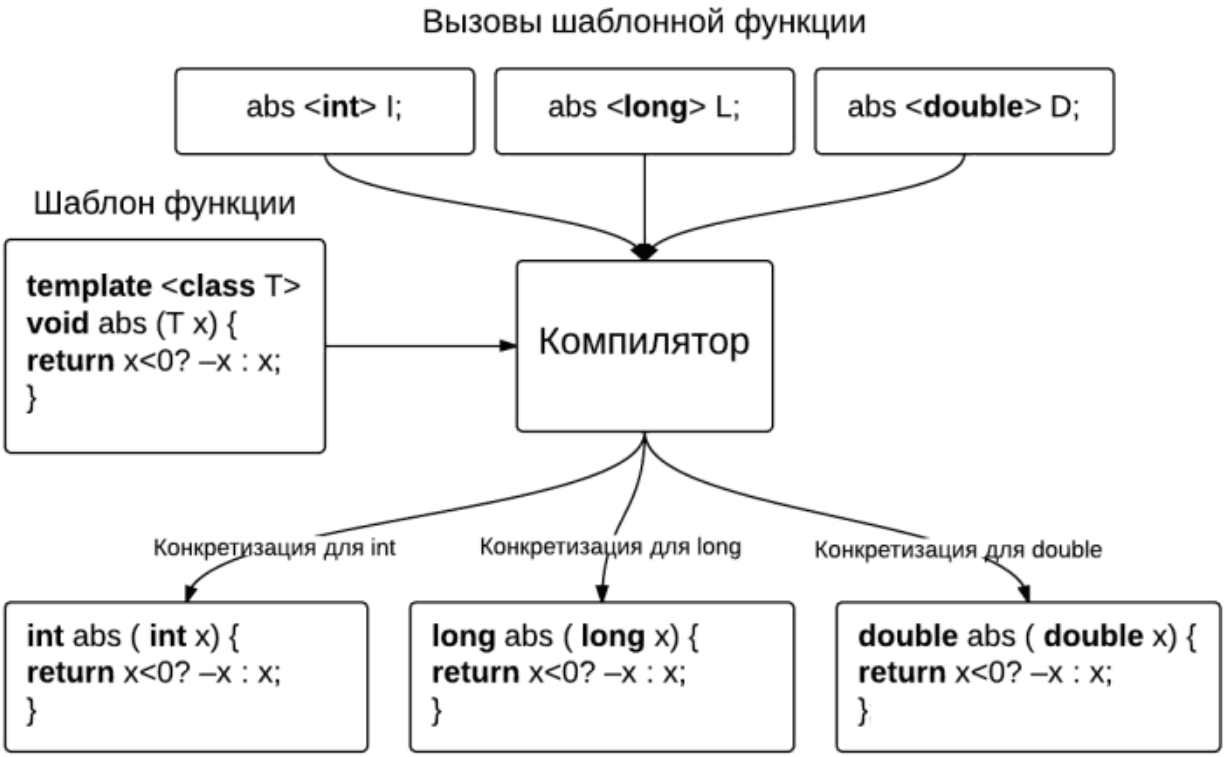
1. Список не может быть пустым.

```
template <>
void abs (T x) {return x<0? –x : x;}
```

2. Все параметры шаблона обязательно должны использоваться при описании формальных параметров функции.

```
template <class T1, class T2>
void abs (T1 x) {return x<0? –x : x;}
```

Конкретизация шаблона функции – создание по шаблону функции, её конкретной реализации для конкретных типов входных параметров её вызова



Явная конкретизация шаблона функции – это явное указание компилятору типа параметра шаблона, когда компилятор не может распознать тип, для которого он должен конкретизировать функцию.

```
template <class T>
T max (T a, T b) {return a>b? a : b;}

float x=2.3, y;
y=max(1000,x);  /* ошибка компилятора: типы значений входных параметров различны, хотя в определении шаблона они имеют одинаковый тип T, поэтому компилятор в замешательстве, он не знает каким из этих двухтипов (float или int) конкретизиро-вать.*/
y=max<float>(1000,x); // явная конкретизация
```

Специализация – это создание программистом отдельной реализации функции для определенных типов данных. Она выполняется программистом, когда компилятор не способен построить функцию по шаблону.

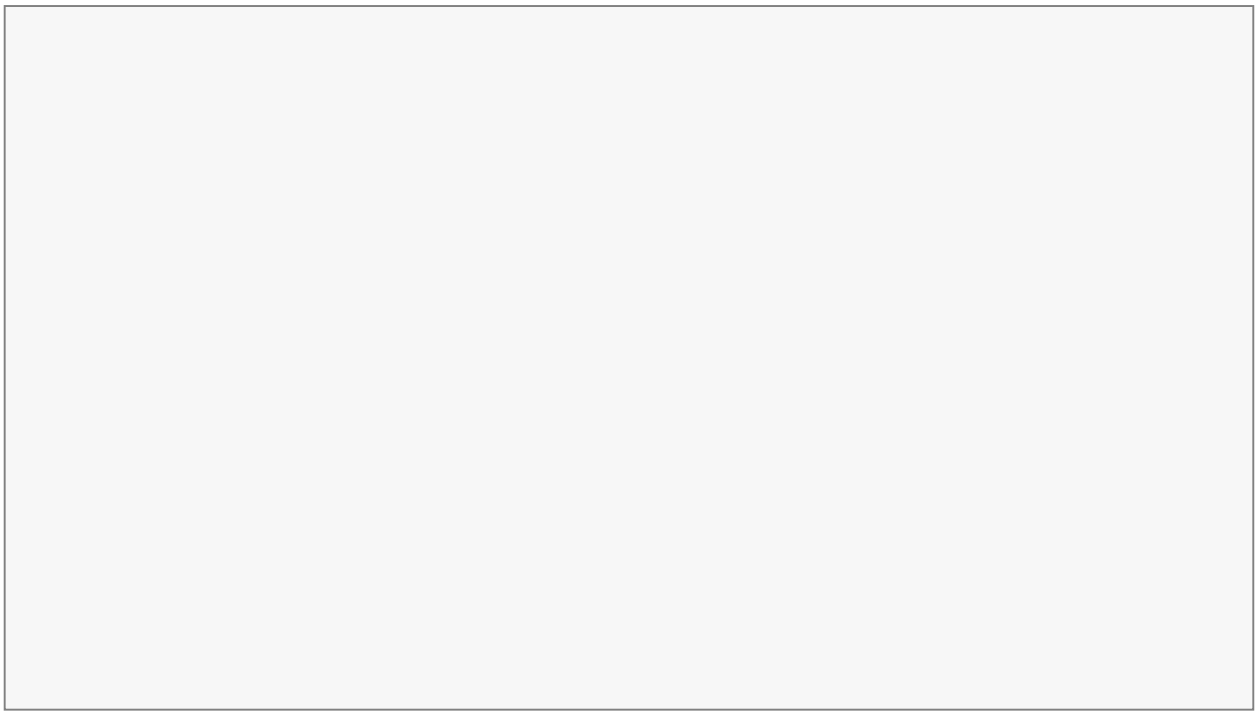
```
template <class T>
T max (T a, T b) {return a>b? a : b;}

//конкретизация для типа строка (char*), т.к. шаблон не пригоден для строк.
char *max (char *a, char *b) {return strcmp(a,b)<0? a : b;}
```

Задания:

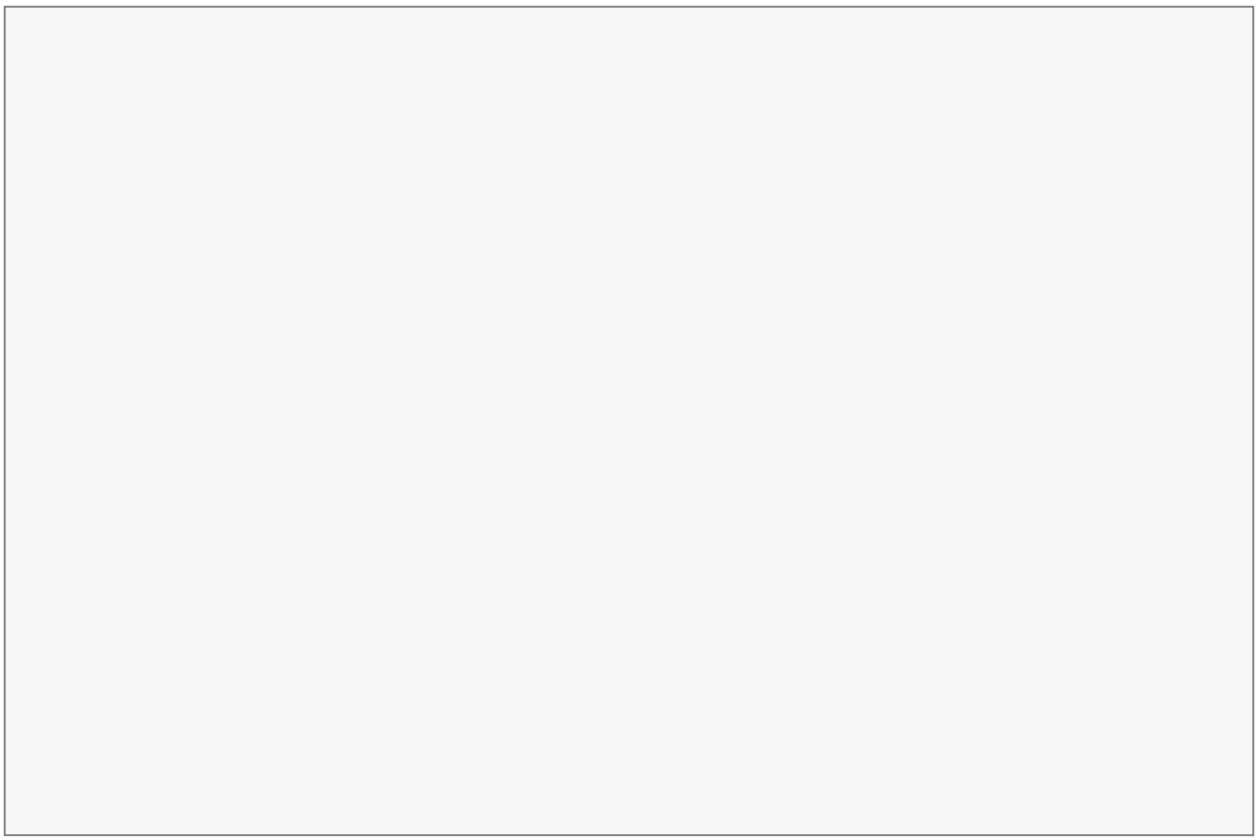
1.Функция:

```
void frukt (int g){
   static int kol=0;
   kol+=g;
};
```

Считает количество (kol) целых (**int**) яблок. Напишите шаблон, благодаря которому можно будет считать не только целые яблоки, но и их кусочки (**float**)



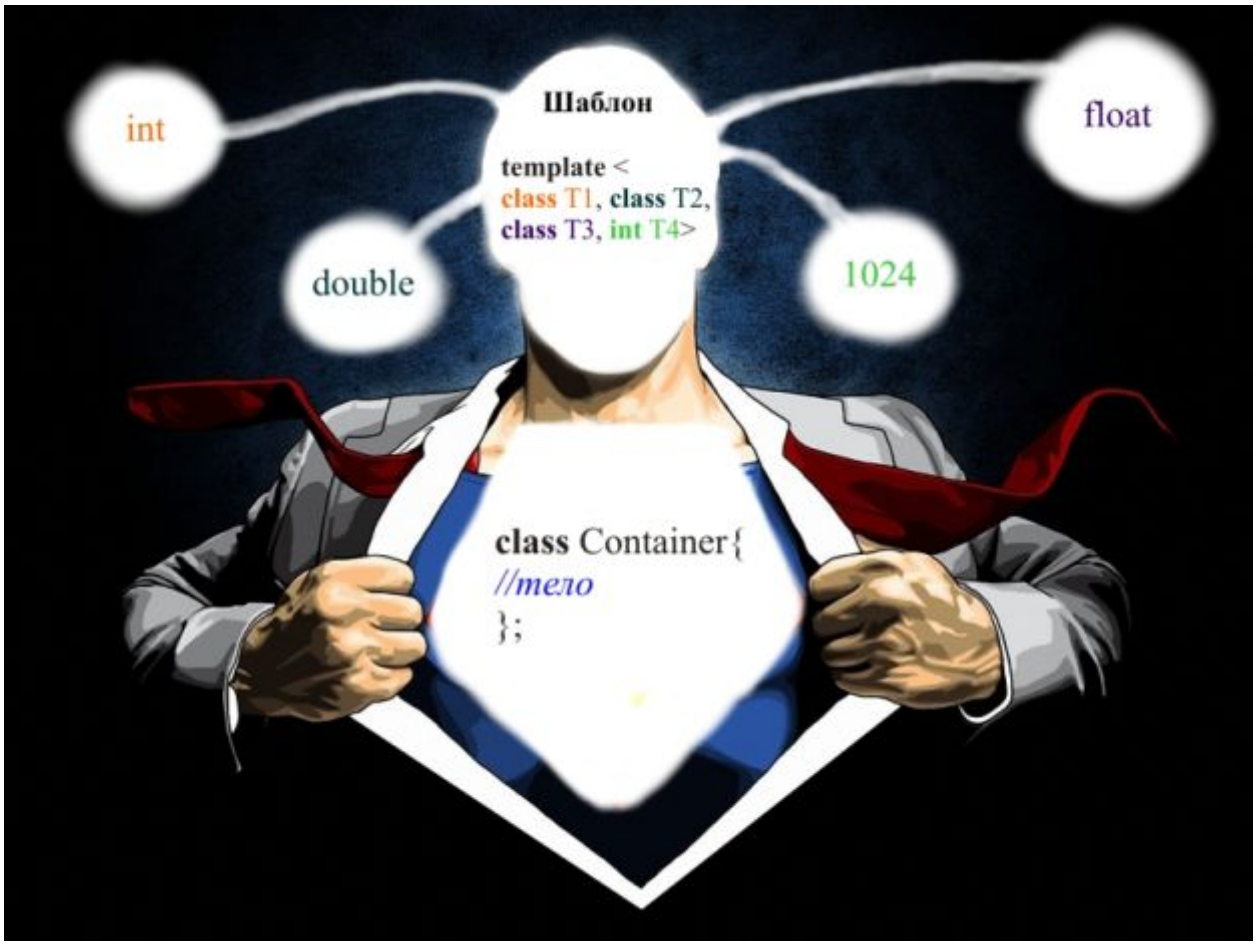
2. На основе задания 1 сделайте специализацию для строк (типа **char*)**, благодаря которой мы сможем считать количество букв в названиях сортов яблок.



Можешь нарисовать что-нибудь с помощью шаблона линий – лекала.

Шаблоны классов – это предписание, по которому компилятор создаёт класс для заданных входных параметров.

Пример шаблона использующего несколько входных параметров:




```
template <class Type, int size>
class Buffer {//тело};
```

Значения по умолчанию:

Для параметров шаблона классов могут быть указаны значения по умолчанию, которые будут использоваться при отсутствии в вызове. В качестве такого аргумента следует выбирать тип или значение, подходящее для большинства конкретизаций. Например, если при вызове шаблона класса **Buffer** не указан размер буфера, то по умолчанию принимается 1024:

```
template <class Type, size = 1024>
class Buffer {//тело};
```

Параметры со значениями по умолчанию должны располагаться в конце списка параметров шаблона.

```
template <class Type, size = 1024>  /* правильно: аргументы по умолчаниюнаходят-ся в конце*/
class Buffer{//тело};
```

```
template <class Type=string, int size> /*неправильно: после параметра по умол-чанию располагается обычный параметр*/
class Buffer{//тело};
```

При определении метода шаблона классов вне класса, в его заголовок добавляется **template** <*параметр_шаблонов_классов*>, а вместо имени класса пишется его уточнённое имя **Point<T>**

```
// Версия шаблона классов Point с внешним определением метода Show()
template <class T>
class Point {
  public:
      void Show();
};

template <class T>
void Point<T> :: Show(){//тело Show()};
```

Задания:

1. Найдите ошибочные объявления (или пары объявлений) шаблонов классов:

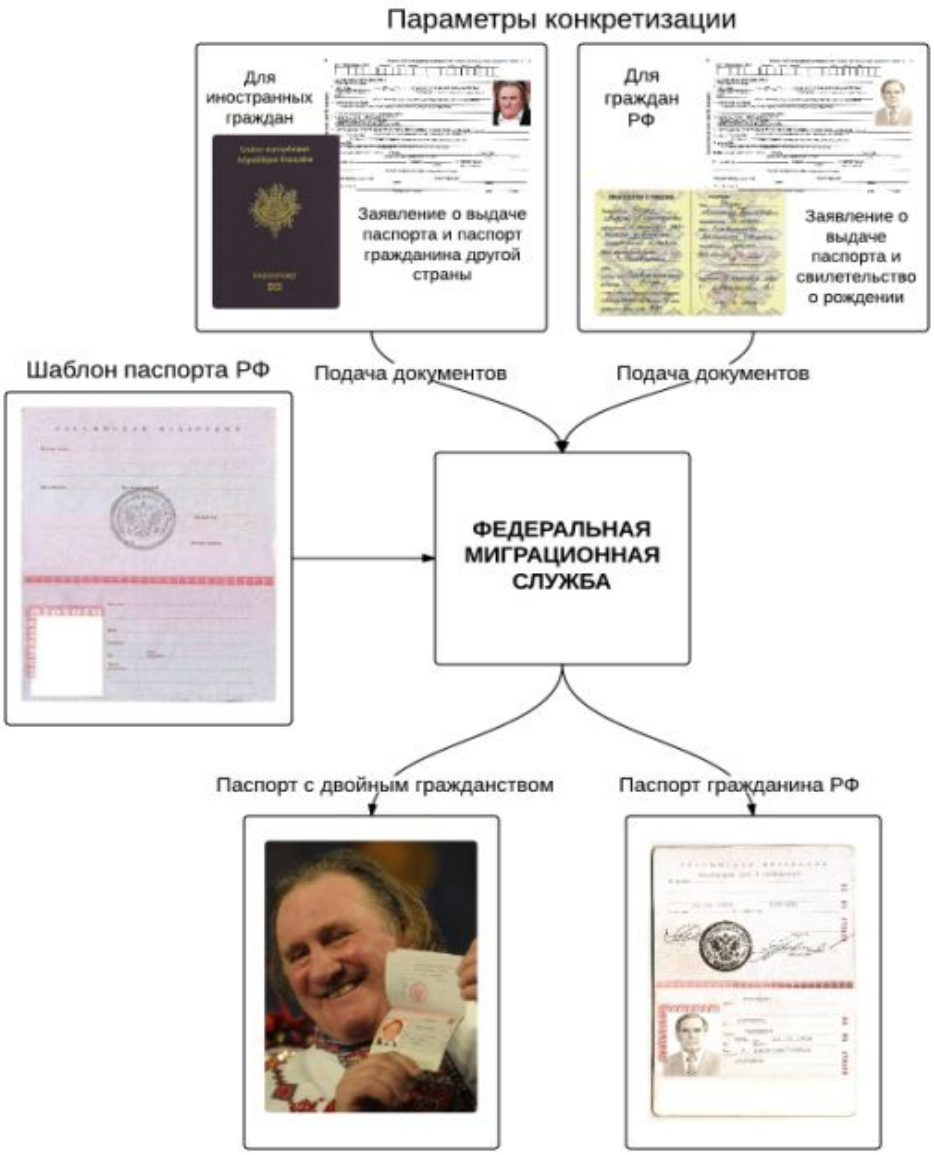
```
(a) template <class Type>
      class Container1;
    template <class Type, int size>
      class Container1;
(b) template <class T, U, class V>
      class Container2;
(c) template <class C1, typename C2>
      class Container3 {};
(d) template <typename myT, class myT>
      class Container4 {};
(e) template <class Type, int *pi>
      class Container5;
(f) template <class Type, int val = 0>
      class Container6;
    template <class T = complex<double>, int v>
      class Container6;
```

2. Создайте класс на основе приведённого выше правильного примера в котором опишите метод с внешним определением.

Наряди девушку.

Конкретизация шаблона классов – это создание на основе шаблона классов компилятором, класса, для определенных параметров (типов и констант).

Алгоритм получения паспорта РФ для граждан РФ и граждан с двойным гражданством:



Конкретизация происходит при создании объекта класса. В имени конкретизируемого шаблона аргументы задаются явно. В следующем примере создается объекта **qi** шаблонного класса **Queue**:

```
Queue<int> qi;   // конкретизируется Queue<int>
```

Конкретизация шаблона классов выполняется компилятором, который значения параметров шаблона вставляет в параметры-заглушки во всём тексте шаблона классов.

```
template <class int>
class Queue {
  public:
    void add( const int & );
  private:
    QueueItem<int> *front;
};
```

Конкретизированный экземпляр шаблона будет иметь уточнённое имя, включающее параметры конкретизации, **Queue<int>** или **Queue<string>**, где **<int>** и **<string>**, следующие за именем **Queue** – это фактические аргументы шаблона.

```
Queue qs;              // ошибка: не указан параметр конкретизации <тип>
```

При конкретизации на основе шаблона классов получается класс, который зетем в программме используется через его уточненное имя.

имя_класса_шаблонов <**параметры_конкретизации**>;

К примеру возможно создать объект шаблона классов **Queue** для различных типов данных:

```
Queue<double> eqd;
Queue<int> lgt;
*pqi = new  Queue<int>;
Queue<int> aqi[1024];
```

Использование расширенного имени при объявлении параметров, локальных переменных:

```
void foo( Queue<int> &qi ) {Queue<int> *pq = &qi;}
```

Задания:

1.Укажите, какие из данных конкретизированных шаблонов действительно приводят к конкретизации:

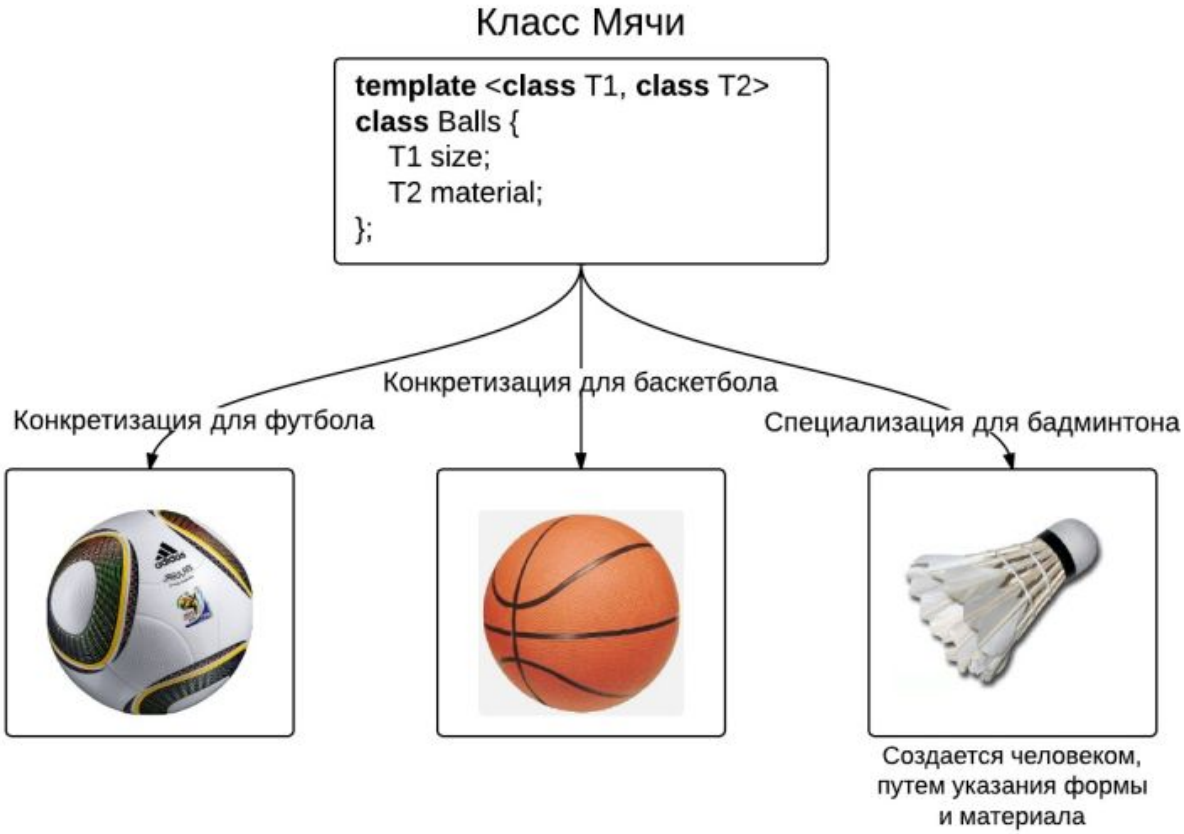
```
template < class Type >
  class Stack { };

void f1( Stack<  char > );  // (a)

class Exercise {
  // ...
  Stack< double > &rsd;    // (b)
  Stack<  int >    si;     // (c)
};

int main() {
  Stack<  char > *sc;      // (d)
  f1( *sc );               // (e)

int iObj = sizeof( Stack<  string > );  // (f)
}
```

2.Какие из следующих конкретизаций шаблонов корректны? Почему?

```
template < int *ptr > class Ptr ( ... };
template < class Type, int size > class Fixed_Array { ... };
template <  int hi, int wid > class Screen { ... };

(a) const int size = 1024;
    Ptr<  &size>bp1;
(b) int arr[10];
    Ptr< arr > bp2;
(c) Ptr < 0 > bp3;
(d) const int hi = 40;
    const int wi = 80;
    Screen<  hi, wi+32 > sObj;
(e) const int size_val = 1024;
    Fixed_Array< string, size_val > fa1;
(f) unsigned int fasize = 255;
    Fixed_Array< int, fasize > fa2;
(g) const double db = 3.1415;
    Fixed_Array< double, db > fa3;
```

Все эти матрешки созданы с помощью шаблонов. Создай свою сам.

Специализация шаблона классов – это создание программистом реализации шаблона классов для конкретных значений его параметров. Необходима, если шаблон классов не пригоден для конкретизации определенным типом данных или, если конкретизация шаблона классов не эффективна по реализации.




```
template <>
class имя_класса <имя_специализированного_типа> {// ...};
```

Например:

```
// общий шаблон
template <class A>
class C {
    A name;
};

// специализация для char*
template <>
class C <char*> {
    char* name;
};
```

Если шаблон классов имеет несколько параметров, то возможна *частичная специализация* – это специализация программистом реализации класса, при которой остаются не специализированные параметры.

Например:

```
// общий шаблон
template <class T1, class T2>
class Pair {
      T1 pole1;
      T2 pole2;
};

//частичная специализация,  где для T2 установлен тип int
template <class T1>
class Pair <T1, int> {
      T1 pole1;
      int pole2;
};
```

Иногда есть смысл специализировать не весь класс, а какой-либо из его методов.

Например:

```
// обобщенный метод
template <class T>
  class A {
     public:
        T min(T a, T b){
           return b < a ? b : a;
        }
};

template <class T>
T A<T>:: min (T a, T b) {return b < a ? b : a;}

// специализированный метод
float A <float>::min (float a, float b) { return b < a ? b : a; }
```

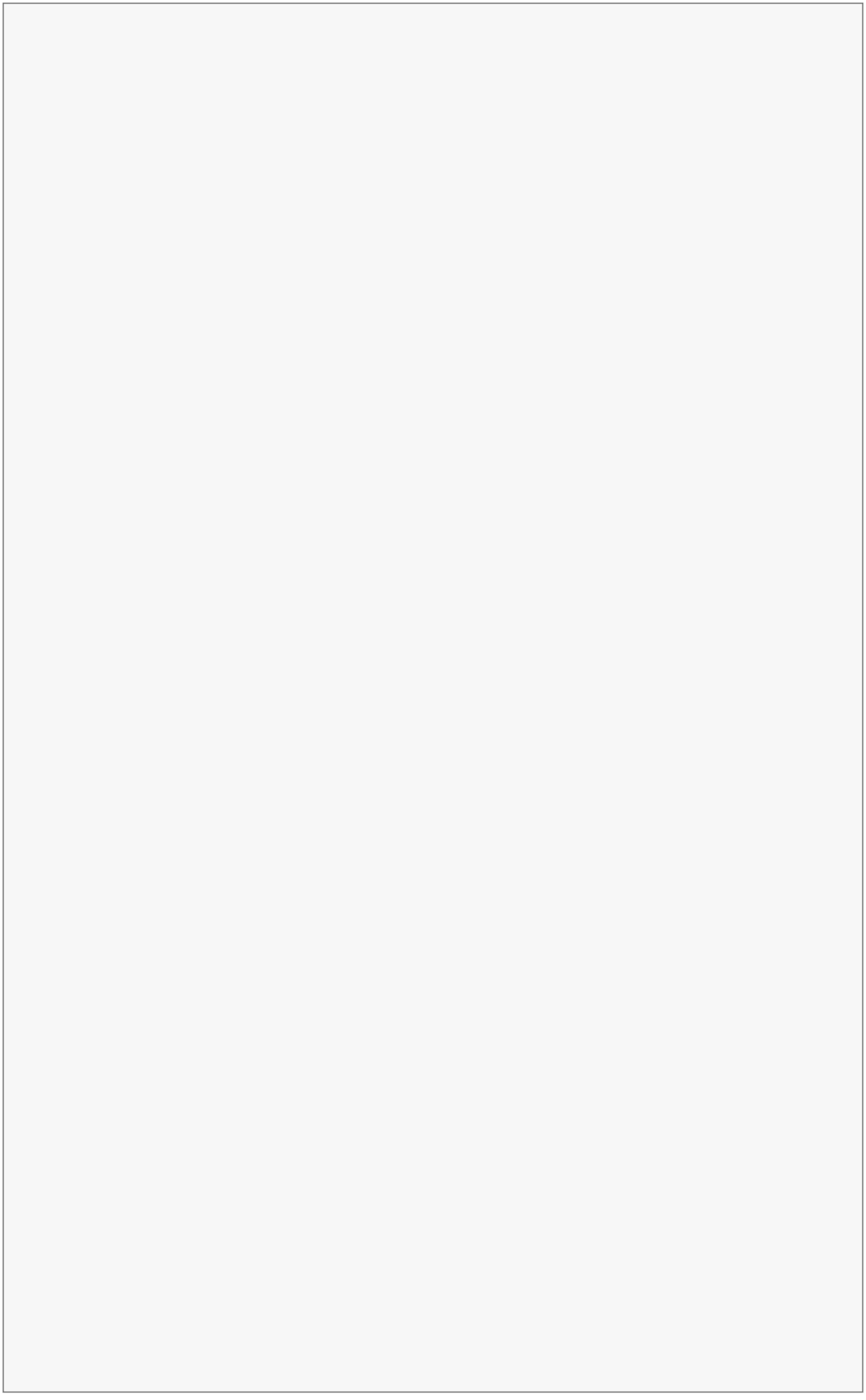
Задание:

Конкретизируйте приведенный ниже шаблон класса **Student** , так, чтобы двоечник-студент, который не смог сдать сессию, отправился служить в воздушно-десантные войска на кухню.

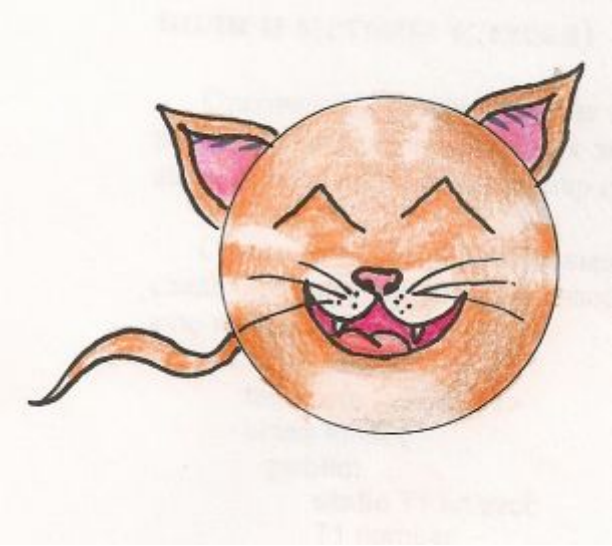

```
template <class T1, int T2, int T3, class T4>
class Student {
        public:
          char Name[100];          // имя студента
          T1 zanytost;             // занятость
          int rod_voisk=T2;        // род войск
          int nom_divizii=T3;      // № дивизии
          T4 mesto_slyjbi;         // место прохождения службы
};
```

enum vid_voisk{
cykchopyt,
vozdysh,
vdv,
morskiye,
};
	enum diviziya{
	106,
	7,
	76,
	};
		class Slyjba{};	**class** Ycheba{};	**class** Rabota{};

class Kuchnya{};	**class** Klub_Ork{};	**class** Sklad{};	**class** Rem_rota{};	**class** Shtab{};



Примени фантазию и измени шаблон.



Статические элементы шаблона классов (статические поля и методы класса)

Статические переменные не исчезают, когда содержащая их функция закончит свою работу, компилятор хранит их значения от одного вызова функции до другого. Каждый конкретизированный экземпляр имеет собственный набор таких членов.

Один статический член совместно используется всеми объектами данного экземпляра, следовательно, все объекты класса **Komanda<int>** совместно используют одну статическую переменную **kolsvoi**.

```
template <class T1>
class Igrok {
  public:
      static T1 kolsvoi;                  // количество игроков команды
      T1 number;                            // номер участника
      static char name[11] [100];         // имена игроков команды
      char trener[100];                    // имя тренера
};
```

Футбольная команда:

Бавария (футбольный клуб, Мюнхен)

Инициализации статического поля выполняется вне класса программист должен использовать следующим образом:

```
template <class T1>
T1 имя_класса <T1>::название_поля=число;
```

Для команды Arsenal:

```
Igrok <float> Arsenal;
template <class T1>  T1 Komanda <T1>::kolsvoi=0; // обнулили количество игроков
```

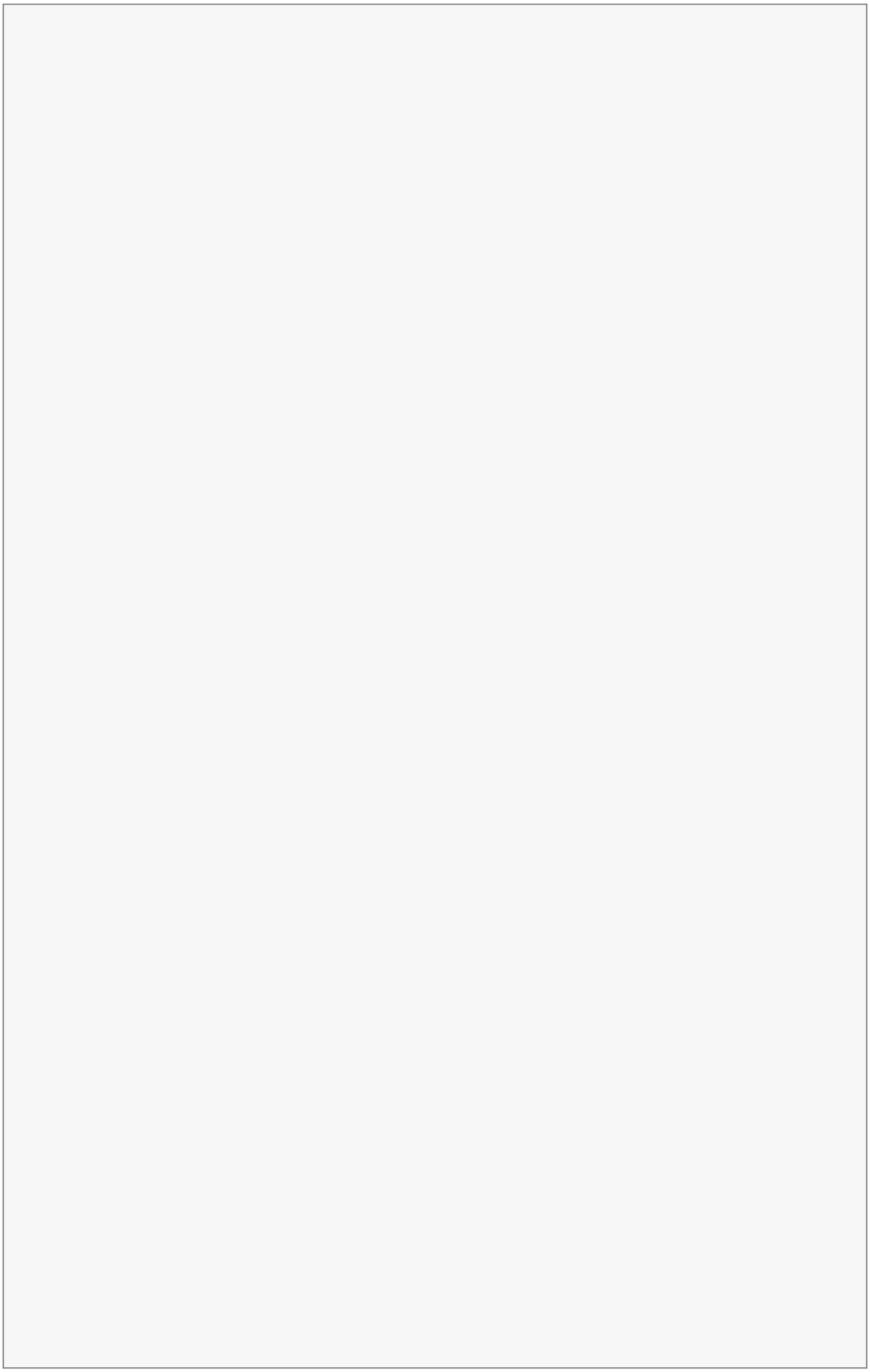
Изменение, статической переменной:

```
main(){
   Igrok <int> Bavariy;
   Komanda<int>:: kolsvoi+=11;        // ФК Бавария теперь состоит из 11 человек
   cout<< Bavariy. Kolsvoi;            // 11
   cout<< Arsenal. Kolsvoi;            // 0
};
```

Задание:

```
template <class T1>
class Igrok {
  public:
      static T1 kolsvoi;                  // количество игроков команды
      T1 number;                           // номер участника
      static char name[11] [100];         // имена игроков команды
      char trener[100];                    // имя тренера
      T1 godosn;                           // год основания ФК
      char gorod[100];                     // имя команды соперника
};
```

Измените объявление полей**: trener**, **godosn**, **gorod** так, чтобы они стали доступны всем игрокам команды. Для понравившейся команды проинициализируйте поля **trener, godson** внутри **main(),** а поле **gorod** вне класса. Нужную дополнительную информацию найдите в интернете.



Изобрази то, что ты понял.